AUEB M.Sc. in Data Science

Course: Social Network Analysis

Homework: 2

Teacher: K. Papakonstantinopoulou

Author: Spiros Politis (p3351814)

Part 1

In this part of the homework, the purpose is to familiarize ourselves with executing *Apache Giraph* map-reduce jobs on *Hadoop*.

To this end, we followed the instructions provided in *hw2.pdf* following the workflow mentioned below:

- Starting the provided VM in *VirtualBox*
- Starting the *distributed file system* and *map reduce* processes by executing the scripts *start-dfs.sh* and *start-mapred.sh*, respectively.
- Executing the command:

```
/usr/local/hadoop/bin/hadoop jar /home/sna/eclipse/workspace/giraph/giraph-examples/target/giraph-examples-1.1.0-for-hadoop-1.2.1-jar-with-dependencies.jar org.apache.giraph.GiraphRunner org.apache.giraph.examples.SimpleShortestPathsComputation -vif org.apache.giraph.io.formats.JsonLongDoubleFloatDoubleVertexInputFormat -vip /user/hduser/input/tiny_graph.txt -vof org.apache.giraph.io.formats.IdWithValueTextOutputFormat -op /user/hduser/output/shortestpaths -w 1
```

The produced result is the following:

0	1.0
1	0.0
2	2.0
3	1.0
4	5.0

As per the instructions of the homework, the output file was renamed with the *Hadoop* task id which, in our case, is *task_201912041554_0001_m_000000*.

Part 2

In this part of the homework, the goal is to implement a *Apache Giraph* computation for finding words in a Boggle board.

For executing the program, we opted for doing so from within *Eclipse* as building the project with *Maven* proved challenging.

Several changes had to be made in class *gr.aueb.sna.GiraphAppRunner* so as to support execution of the *org.apache.giraph.examples.SimpleBoggleComputation*. These changes are outlined below:

- Change of the *Apache Giraph* input format:

```
giraphConf.setVertexInputFormatClass(BoggleInputFormat.class);
```

- Change of the *Apache Giraph* computation class:

```
giraphConf.setComputationClass(SimpleBoggleComputation.class);
```

- Checking whether the output path already exists in *Hadoop* and, if yes, delete it:

```
// Delete Hadoop output path if exists.
FileSystem hdfs = FileSystem.get(conf);
if (hdfs.exists(new Path(getOutputPath()))) {
    hdfs.delete(new Path(getOutputPath()), true);
}
```

For the actual computation, a short description of the algorithm followed is outlined below:

- In Superstep 0, broadcast the vertex letter and Id to all connected edges.

```
    // Every vertex should send an initial message to all its neighbors
with
    // a word starting with its respective letter, and the vertex id (so
    // that the same vertex is not used again in this word).
    if (getSuperstep() == 0) {
        TextArrayListWritable talw = new TextArrayListWritable();
        talw.add(new Text(vertex.getId().toString().substring(0, 1)));
        talw.add(vertex.getId());

        sendMessageToAllEdges(vertex, talw);
    }
```

- In every Superstep other than 0, we read the messages passed in each vertex:

```
    // Read messages.
        for (TextArrayListWritable message : messages) {
        ...
    }
```

- If an exact match of a word in the dictionary is found, we place the text occured from concatenation of the vertex text with the received text as the vertex value:

```
    // If the word is in the dictionary, the vertex should add the
    // word to the value of the vertex.
    // Note that the type value of the vertex is defined as a list
    // of Text objects.
    // Examine if the word is contained in the dictionary.
    if (dictionary.contains(concatenatedText.toString())) {

        // Add the word to the value of the vertex.
        vertex.getValue().add(concatenatedText);
    }
```

- If a word starting with any word in the dictionary is found, we multicast the text occured from concatenation of the vertex text with the received text to edges of the vertex that have not received it previously:

```
    TextArrayListWritable talw = new TextArrayListWritable();
    talw.add(concatenatedText);
    talw.add(vertex.getId());

    // Send message to vertices that have not received it previously.
    HashSet<Text> unvisitedEdges = this.getUnvisitedEdges(vertex, message);
    sendMessageToMultipleEdges(unvisitedEdges.iterator(), talw);
```

The output of the process is in file *part-m-00000* in path *boggle_output* and is provided below:

S:3:2	[GEEKS]
I:1:2	[]
Q:3:1	[]
E:2:2	[]
E:3:3	[]
K:2:3	[]
G:1:1	[]

Z:1:3	[QUIZ]
U:2:1	[]

Also, included is the execution log provided in file *log.txt*. The file includes *info* level logging output.